Python 数据分析与数据挖掘（Python for Data Analysis&Data Mining）

Chap 7 电力窃漏电用户自动识别

内容：

- 对分析和挖掘应用目标的分析
- 将生产数据转换为系统能够处理的数据类型
- 数据预处理、缺失值
- 数据变换和归一化
- 算法：决策树算法（DT），朴素贝叶斯算法的性能对比
- 应用领域：信用卡欺诈、保险欺诈、征信、风控、偷漏税等

实践：

- 实例1：电力窃漏电用户自动识别（Python数据分析与挖掘实战 第六章）
- 实例2：天气数据预处理

安装sklearn库，应用sklearn库中的机器学习算法，比较决策树、朴素贝叶斯算法的性能。

准备工作：导入库，配置环境等

In [1]:
```
# 必要准备工作：导入库，配置环境等
from __future__ import division

import numpy as np
import pandas as pd
np.set_printoptions(precision=4, suppress=True)

# 启动绘图
%matplotlib inline
import matplotlib.pyplot as plt
```

实例1：电力窃漏电用户自动识别

1. 问题背景

传统防窃漏电方法主要通过定期巡检、定期校验电表、用户举报等方法来发现窃电或计量装置故障。 缺点 ： 对人的依赖性太强，抓窃查漏的目标不明确。

目前主要通过营销稽查人员、电力检查人员和计量工作人员利用计量异常报警功能和电能量数据查询功能进行用户用电情况的在线监控，采集电量异常、负荷异常、终端报警、主站报警、线损异常等信息，建立数据分析模型，来实时电测窃漏电情况和发现计量装置的故障。根据报警前后时间客户计量点有关的电流、电压、负荷数据情况等，构建不同指标加权的异常分析模型，实现客户是否存在窃电、违章用电和计量装置故障等。 缺点 ： 终端误报或漏报过多，无法真正快速精确定位窃漏

电用户。而且，各输入指标权重的确定需要专家的知识和经验来判断，具有很大的主观性，存在明显的缺陷，效果不尽如人意。

现有的电力计量自动化系统能够采集到各相电流、电压、功率因数等用电负荷数据以及用电异常等终端报警信息。从这些数据信息中提取出窃漏电用户的关键特征，构建窃漏电用户的识别模式，就能自动检查、判断用户是否存在窃漏电行为。

2. 原始数据

1. 用户的用电负荷数据

采集时间间隔为15分钟，即0.25小时，可以进一步计算用户的用电量。

表6-1 某企业大用户用电负荷数据

用户编号	时间	有功总	B相	C相	电流A相	电流B相	电流C相	电压A相	电压B相	电压C相	功率因数	功率因数A	功率因数B	功率因数C
0319001000019011001	2011/11/10	202	0	349.2	33.6	0	33.4	10500	0	10500	0.784	0.573	-10000	0.996
0319001000019011001	2011/11/10 0:15	194.8	0	355.4	32.4	0	34	10500	0	10500	0.789	0.573	-10000	0.996
0319001000019011001	2011/11/10 0:30	210.4	0	366	35	0	35	10500	0	10500	0.784	0.573	-10000	0.996
0319001000019011001	2011/11/10 0:45	199.6	0	376.4	33.2	0	36	10500	0	10500	0.793	0.573	-10000	0.996
0319001000019011001	2011/11/10 1:00	191.2	0	334.6	31.8	0	32	10500	0	10500	0.785	0.573	-10000	0.996
0319001000019011001	2011/11/10 1:15	192.4	0	340.8	32	0	32.6	10500	0	10500	0.786	0.573	-10000	0.996
0319001000019011001	2011/11/10 1:30	192.4	0	353.4	32	0	33.8	10500	0	10500	0.79	0.573	-10000	0.996
0319001000019011001	2011/11/10 1:45	197.2	0	357.6	32.8	0	34.2	10500	0	10500	0.789	0.573	-10000	0.996
0319001000019011001	2011/11/10 2:00	178	0	320.8	29.6	0	30.4	10500	0	10600	0.788	0.573	-10000	0.996

2. 用户的终端报警信息

与窃漏电相关的报警能用于识别用户的窃漏电行为

3. 用户违约、窃电处理通知书

记录用户的用电类别和窃电时间

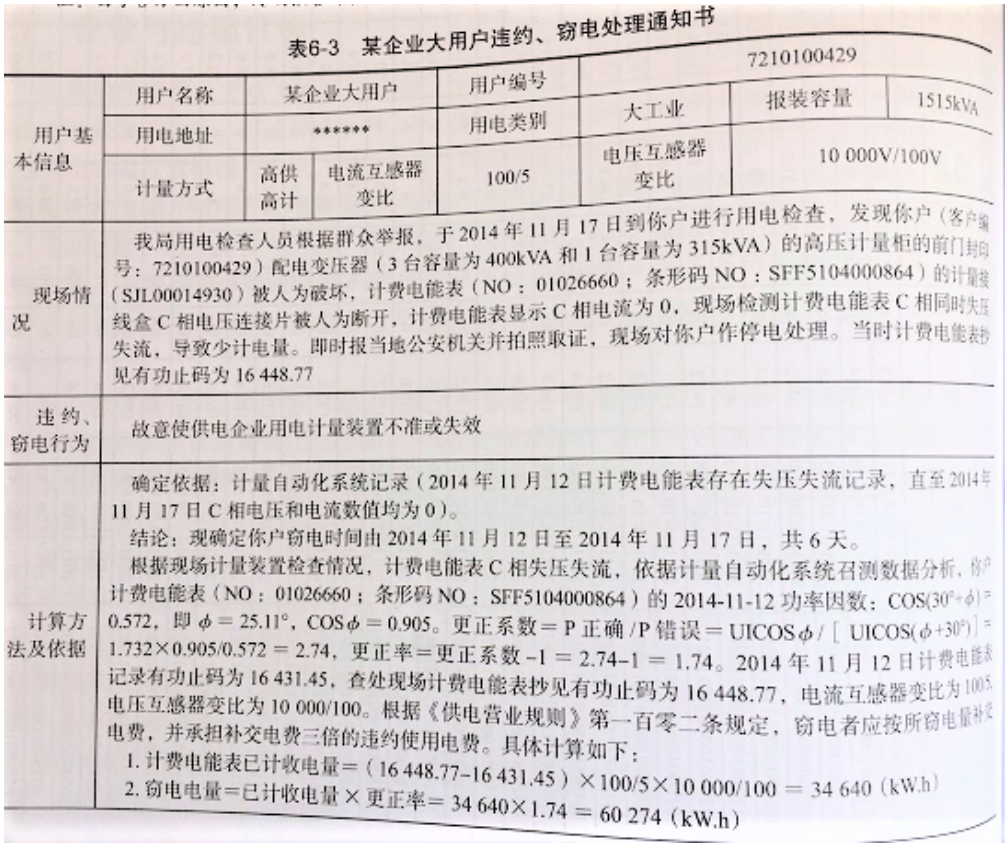


表6-3 某企业大用户违约、窃电处理通知书

用户基本信息	用户名称	某企业大用户		用户编号	7210100429	
	用电地址	******		用电类别	大工业	报装容量 1515kVA
	计量方式	高供高计	电流互感器变比	100/5	电压互感器变比	10 000V/100V
现场情况	我局用电检查人员根据群众举报，于2014年11月17日到你户进行用电检查，发现你户（客户编号：7210100429）配电变压器（3台容量为400kVA和1台容量为315kVA）的高压计量柜的前门封印（SJL00014930）被人为破坏，计费电能表（NO：01026660；条形码NO：SFF5104000864）的计量接线盒C相电压连接片被人为断开，计费电能表显示C相电流为0，现场检测计费电能表C相同时失压失流，导致少计电量。即时报当地公安机关并拍照取证，现场对你户作停电处理。当时计费电能表抄见有功止码为16 448.77					
违约、窃电行为	故意使供电企业用电计量装置不准或失效					
计算方法及依据	确定依据：计量自动化系统记录（2014年11月12日计费电能表存在失压失流记录，直至2014年11月17日C相电压和电流数值均为0）。 结论：现确定你户窃电时间由2014年11月12日至2014年11月17日，共6天。 根据现场计量装置检查情况，计费电能表C相失压失流，依据计量自动化系统召测数据分析，你户计费电能表（NO：01026660；条形码NO：SFF5104000864）的2014-11-12功率因数：$COS(30°+\phi)=0.572$，即$\phi=25.11°$，$COS\phi=0.905$。更正系数＝P正确/P错误＝$UICOS\phi/[UICOS(\phi+30°)]=1.732\times0.905/0.572=2.74$，更正率＝更正系数-1＝2.74-1＝1.74。2014年11月12日计费电能表记录有功止码为16 431.45，查处现场计费电能表抄见有功止码为16 448.77，电流互感器变比为100/5，电压互感器变比为10 000/100。根据《供电营业规则》第一百零二条规定，窃电者应按所窃电量补交电费，并承担补交电费三倍的违约使用电费。具体计算如下： 1. 计费电能表已计收电量＝（16 448.77-16 431.45）×100/5×10 000/100＝34 640（kW.h） 2. 窃电电量＝已计收电量×更正率＝34 640×1.74＝60 274（kW.h）					

3. 建模目标

- 1） 分析数据，归纳出窃漏电用户的关键特征 -- 数据分析和特征抽取
- 2） 构建建窃漏电用户的识别模型 -- 构建模型
- 3） 针对实时监测数据，利用窃漏电用户识别模型实现实时监控 -- 使用模型

4. 分析方法与过程

- 1）选择性地抽取与窃漏电行为相关的原始数据。
- 2）窃漏电用户占总用户的很小部分，同时某些大用户不大可能存在窃漏电行为（如银行、税务、学校、行政机构等）。因此，预处理时将这类用户剔除。
- 3）总用电量不能直接体现出用户的窃漏电行为，终端报警存在很多误报和漏报情况。因此，需要结合历史窃漏电用户信息，总结窃漏电用户的行为规律，从数据中提炼出描述窃漏电用户的特征。
- 4） 对样本数据进行预处理，包括数据清洗、缺失值和数据变换。
- 5） 构建训练数据集
- 6） 构建窃漏电用户识别模型。
- 7） 针对实时监测数据（即测试数据），利用模型进行实时监控预测。

窃漏电用户识别流程图包括以下步骤

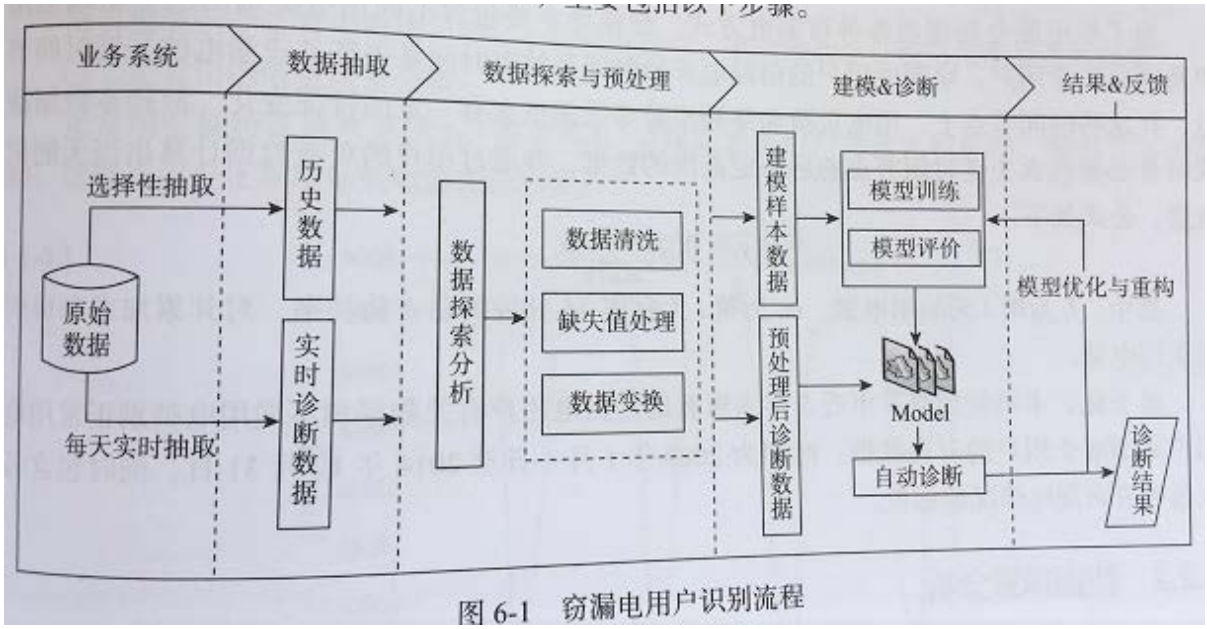


图 6-1 窃漏电用户识别流程

5. 数据分析

5.1 分布分析

对5年所有的窃漏电用户进行分布分析，统计各类型用户的窃漏电分布情况。

5.2 周期性分析

对比正常用户和窃漏电用户的用电量情况。

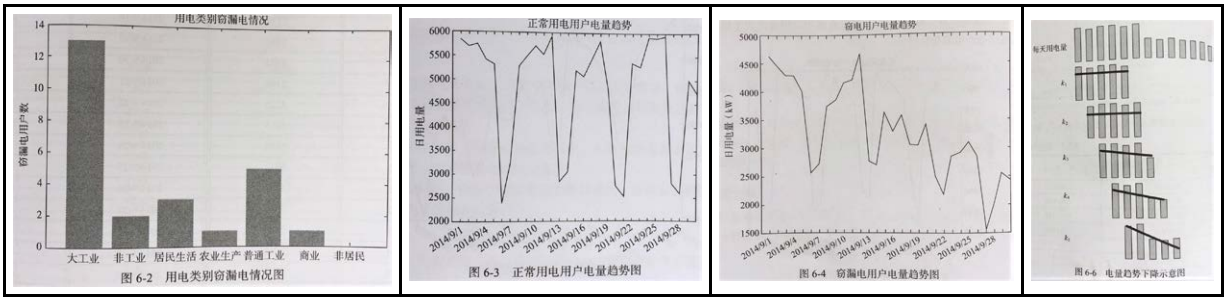


图 6-2 用电类别窃漏电情况图 图 6-3 正常用电用户电量趋势图 图 6-4 窃漏电用户电量趋势图 图 6-6 电量趋势下降示意图

数据分析和探索结果：

- 非居民类别的用户不存在窃漏电情况，可以不考虑
- 正常用户的用电量比较平稳，没有太大的波动
- 窃漏电用户的用电量有明显下降的趋势，这个可以作为用户异常用电的电量指标特征。

怎么表征这个特征？

- 随着时间推移，在各个统计窗口对用电量做直线拟合的斜率，可以看到斜率随着时间逐步下降。
- 对统计当天设定前后5点为统计窗口，计算这11天内的电量趋势下降情况。how？
  - 首先计算这11天中每天的用电量和用电趋势，第i天的用电趋势是考虑前后5天的用电量斜率
  - 如果当天i比前一天用电量减少，则当天i的下降指标为 D(i)=1；否则， D(i)=0
  - 11天内的电量趋势下降指标为：这10天所有下降指标为 D(i)的累加和

此外，还要考虑线损指标（衡量供电线路的损失比例，计算用户所属的线路在当天的线损率）。

与窃漏电相关的终端报警类型有电压缺相、电压断相、电流反极性等告警，计算各种报警类型与窃

漏电相关的终端报警的次数和，作为报警类指标。

最终，构建的数据集中包含三类数据特征，数据特征和样本数据如下图：

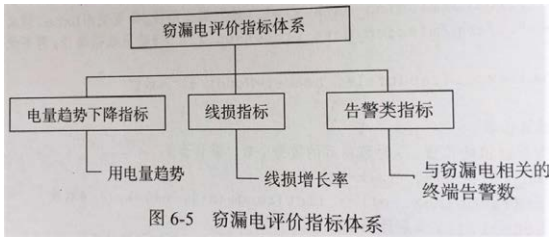


图 6-5 窃漏电评价指标体系

表6-8 专家样本数据

时 间	用户编号	电量趋势下降指标	线损指标	告警类指标	是否窃漏电
2014年9月6日	9900667154	4	1	1	1
2014年9月20日	9900639431	4	0	4	1
2014年9月17日	9900585516	2	1	1	1
2014年9月14日	9900531154	9	0	0	0
2014年9月17日	9900491050	3	1	0	0
2014年9月13日	9900461501	2	0	0	0
2014年9月22日	9900412593	5	0	2	1

6. 构建识别模型

6.1 数据划分

从样本数据集中抽取 20% 做测试（用于评估模型），剩下 80% 做训练数据（用于构建模型）

In [4]:
```
#-*- coding: utf-8 -*-

import numpy as np
import pandas as pd
from pandas import Series, DataFrame

datafile = 'data/powerdata.xls' #数据名
data = pd.read_excel(datafile) #读取数据，数据的前三列是特征，第四列是标签
print len(data)
data.head()
```

```
291
```

Out[4]:

	电量趋势下降指标	线损指标	告警类指标	是否窃漏电
0	4	1	1	1
1	4	0	4	1
2	2	1	1	1
3	9	0	0	0
4	3	1	0	0

In [5]:
```
# 数据划分：20%做测试，剩下80%做训练数据
from random import shuffle #导入随机函数shuffle，用来打乱数据

data = data.as_matrix() #将表格转换为矩阵
shuffle(data) #随机打乱数据

p = 0.8 #设置训练数据比例
train = data[:int(len(data)*p),:] #前80%为训练集
test = data[int(len(data)*p):,:] #后20%为测试集

print len(train), len(test)
```

```
232 59
```

6.2 构建模型系统

1). 决策树模型： 使用**sklearn**中的决策树算法

决策树（Decision Tree）算法的原理 （见Slides）

Accuracy = 0.9322

In [17]:
```
#构建CART决策树模型
from sklearn.tree import DecisionTreeClassifier #导入决策树模型

# 训练模型
tree = DecisionTreeClassifier() # 建立决策树模型
tree.fit(train[:,:3], train[:,3]) # 训练学习模型

# 保存模型
from sklearn.externals import joblib
treefile = 'data/output/treemodel.pkl' # 存放输出的模型
joblib.dump(tree, treefile)
```

Out[17]:
```
['data/output/treemodel.pkl',
 'data/output/treemodel.pkl_01.npy',
 'data/output/treemodel.pkl_02.npy',
 'data/output/treemodel.pkl_03.npy',
 'data/output/treemodel.pkl_04.npy']
```

In [7]:
```
# sklearn使用predict方法直接给出预测结果。
predict_result = tree.predict(test[:,:3])
```

In [10]:
```
# 预测结果
print len(predict_result)
predict_result
```

```
59
```

Out[10]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0], dtype=int64)
```

In [12]:
```
# 模型准确性
accuracy = tree.score(test[:,:3], test[:,3])  # (X,y)
accuracy
```

Out[12]:
```
0.93220338983050843
```

In [16]:
```
# 对比预测结果和test的真实结果
print predict_result
print test[:,3]
```

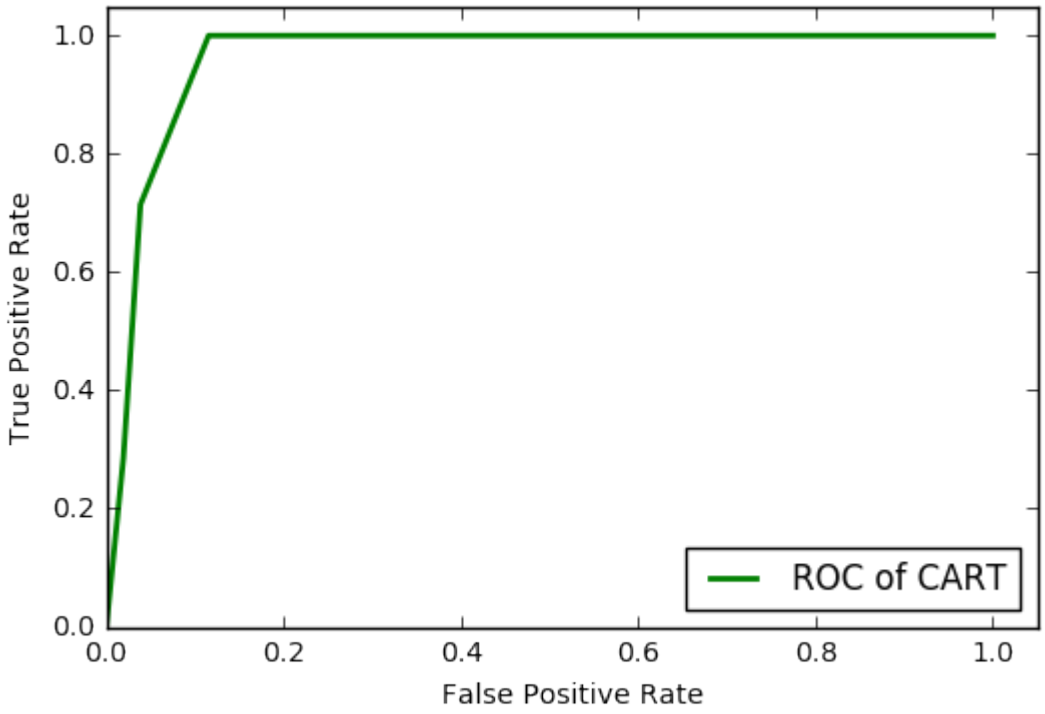
```
[0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
 1
 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
 0
 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
```

结果可视化

In [14]:
```
# 启动绘图
%matplotlib inline
import matplotlib.pyplot as plt
```

In [15]:
```
from sklearn.metrics import roc_curve #导入ROC曲线函数

fpr, tpr, thresholds = roc_curve(test[:,3], tree.predict_proba(test[:,:3
])[:,1], pos_label=1)
plt.plot(fpr, tpr, linewidth=2, label = 'ROC of CART', color = 'green')
#作出ROC曲线
plt.xlabel('False Positive Rate') #坐标轴标签
plt.ylabel('True Positive Rate') #坐标轴标签
plt.ylim(0,1.05) #边界范围
plt.xlim(0,1.05) #边界范围
plt.legend(loc=4) #图例
plt.show() #显示作图结果
```



2). 朴素贝叶斯： 使用**sklearn**中的贝叶斯算法

朴素贝叶斯(Naive Bayes)算法的原理 （见Slides）

Accuracy = 0.8814

In [28]:
```
import numpy as np
import pandas as pd
from random import shuffle #导入随机函数shuffle，用来打乱数据
from sklearn.naive_bayes import MultinomialNB

datafile = 'data/powerdata.xls' #数据名
data = pd.read_excel(datafile) #读取数据，数据的前三列是特征，第四列是标签

# 数据划分：20%做测试，剩下80%做训练数据
data = data.as_matrix() #将表格转换为矩阵
shuffle(data) #随机打乱数据
p = 0.8 #设置训练数据比例
```

```
train = data[:int(len(data)*p),:] #前80%为训练集
test = data[int(len(data)*p):,:] #后20%为测试集

# 构建并训练模型
clf = MultinomialNB()
clf.fit(train[:,:3], train[:,3])

# 使用模型进行预测, sklearn使用predict方法直接给出预测结果。
predict_result = clf.predict(test[:,:3])

# 模型准确性
accuracy = clf.score(test[:,:3], test[:,3])  # (X,y)
accuracy
```

Out[28]: 0.88135593220338981

3). 神经网络模型：

Keras大大简化了搭建各种神经网络模型的步骤。 conda install theano conda install keras 下面的例子是简单搭建一个LSTM,详见代码L03/code/LSTM.py, 输出结果在LSTM-Outputs.txt 输出结果每行为每个测试样本在10个类别中的预测概率分布。 注意： 目前keras中使用到TensorFlow，而TensorFlow目前无法在Windows系统上安装，只支持Mac、Linux系统，因此下面的代码需要在服务器端运行。

注意： Windows系统下目前没有tensorflow，因此使用keras和tensorflow构建的神经网络模型代码必须在Unix服务器端使用

下面代码在 Unix服务器端使用 #-*- coding: utf-8 -*- import pandas as pd from random import shuffle datafile = 'data/powerdata.xls' data = pd.read_excel(datafile) data = data.as_matrix() shuffle(data) p = 0.8 #设置训练数据比例 train = data[:int(len(data)*p),:] test = data[int(len(data)*p):,:] # 构建LM神经网络模型 from keras.models import Sequential #导入神经网络初始化函数 from keras.layers.core import Dense, Activation #导入神经网络层函数、激活函数 netfile = 'data/output/net.model' #构建的神经网络模型存储路径 net = Sequential() #建立神经网络 net.add(Dense(input_dim = 3, output_dim = 10)) #添加输入层（3节点）到隐藏层（10节点）的连接 net.add(Activation('relu')) #隐藏层使用relu激活函数 net.add(Dense(input_dim = 10, output_dim = 1)) #添加隐藏层（10节点）到输出层（1节点）的连接 net.add(Activation('sigmoid')) #输出层使用sigmoid激活函数 net.compile(loss = 'binary_crossentropy', optimizer = 'adam', class_mode = "binary") #编译模型，使用adam方法求解 net.fit(train[:,:3], train[:,3], nb_epoch=1000, batch_size=1) #训练模型，循环1000次 net.save_weights(netfile) #保存模型 predict_result = net.predict_classes(train[:,:3]).reshape(len(train)) #预测结果变形 # '''这里要提醒的是，keras用predict给出预测概率，predict_classes才是给出预测类别， # 而且两者的预测结果都是n x 1维数组，而不是通常的 1 x n''' from cm_plot import * #导入自行编写的混淆矩阵可视化函数 cm_plot(train[:,3], predict_result).show() #显示混淆矩阵可视化结果 from sklearn.metrics import roc_curve #导入ROC曲线函数 predict_result = net.predict(test[:,:3]).reshape(len(test)) fpr, tpr, thresholds = roc_curve(test[:,3], predict_result, pos_label=1) plt.plot(fpr, tpr, linewidth=2, label = 'ROC of LM') #作出ROC曲线 plt.xlabel('False Positive Rate') #坐标轴标签 plt.ylabel('True Positive Rate') #坐标轴标签 plt.ylim(0,1.05) #边界范围 plt.xlim(0,1.05) #边界范围 plt.legend(loc=4) #图例 plt.show() #显示作图结果

实例**2**： 天气数据预处理

天气数据的采集获取已经在进行中

In [13]:
```
import numpy as np
import pandas as pd

f = 'data/weatherall.csv'
data = pd.read_csv(f, index_col='Date', encoding='GBK') # 使用date列作为行索
```

```
引，中文编码
data.index = pd.to_datetime(data.index)  #将字符串索引转换成时间索引
print len(data)
data.head()
```

```
61614
```

Out[13]:

	City	Wkday	Outlook	TempH	TempL
Date					
2017-01-24	平定	1	多云	NaN	-8.0
2017-01-25	平定	2	晴	6.0	-6.0
2017-01-26	平定	3	晴	4.0	-9.0
2017-01-27	平定	4	晴	3.0	-8.0
2017-01-28	平定	5	多云转阴	4.0	-7.0

In [14]:
```
#数据中有缺失值NA
data.fillna(method = 'backfill', inplace=True) #就地使用第二天的值填充
data.head()
```

Out[14]:

	City	Wkday	Outlook	TempH	TempL
Date					
2017-01-24	平定	1	多云	6.0	-8.0
2017-01-25	平定	2	晴	6.0	-6.0
2017-01-26	平定	3	晴	4.0	-9.0
2017-01-27	平定	4	晴	3.0	-8.0
2017-01-28	平定	5	多云转阴	4.0	-7.0

In [15]:
```
data.head()
```

Out[15]:

	City	Wkday	Outlook	TempH	TempL
Date					
2017-01-24	平定	1	多云	6.0	-8.0
2017-01-25	平定	2	晴	6.0	-6.0
2017-01-26	平定	3	晴	4.0	-9.0
2017-01-27	平定	4	晴	3.0	-8.0
2017-01-28	平定	5	多云转阴	4.0	-7.0

In [16]:
```
data['TempL'].tail()
```

Out[16]:
```
Date
2017-02-02    3.0
2017-02-03    7.0
2017-02-04    9.0
2017-02-05    7.0
2017-02-06    3.0
Name: TempL, dtype: float64
```

In [17]:
```
data['TempMean'] = (data['TempL'] + data['TempH']) / 2
data.head()
```

Out[17]:

	City	Wkday	Outlook	TempH	TempL	TempMean
Date						
2017-01-24	平定	1	多云	6.0	-8.0	-1.0
2017-01-25	平定	2	晴	6.0	-6.0	0.0
2017-01-26	平定	3	晴	4.0	-9.0	-2.5
2017-01-27	平定	4	晴	3.0	-8.0	-2.5
2017-01-28	平定	5	多云转阴	4.0	-7.0	-1.5

astype对dataframe中元素，进行类型转换

data['TempL'] = data['TempL'].replace('.','').astype(int)

In [ ]:
```

```